Benchmarking machine-learning-based object detection on a UAV and mobile platform

Martinez-Alpiste, Ignacio; Casaseca-de-la-Higuera, Pablo; Alcaraz-Calero, Jose M.; Grecos, Christos; Wang, Qi

Published in:
2019 IEEE Wireless Communications and Networking Conference (WCNC)

DOI:
10.1109/WCNC.2019.8885504

Published: 30/04/2019

Document Version
Peer reviewed version

Martinez-Alpiste, I., Casaseca-de-la-Higuera, P., Alcaraz-Calero, J. M., Grecos, C., & Wang, Q. (2019). Benchmarking machine-learning-based object detection on a UAV and mobile platform. In 2019 IEEE Wireless Communications and Networking Conference (WCNC) (IEEE Proceedings). IEEE. https://doi.org/10.1109/WCNC.2019.8885504

Benchmarking Machine-Learning-Based Object Detection on a UAV and Mobile Platform

Ignacio Martinez-Alpiste
School of Engineering and Computing
University of the West of Scotland
Paisley, UK
Ignacio.Alpiste@uws.ac.uk

Pablo Casaseca-de-la-Higuera
Laboratorio de Procesado de Imagen
Universidad de Valladolid
Valladolid, Spain
casaseca@lpi.tel.uva.es

Jose Alcaraz-Calero
School of Engineering and Computing
University of the West of Scotland
Paisley, UK
Jose.Alcaraz-Calero@uws.ac.uk

Christos Grecos
Computer Science Department
Central Washington University
Washington, USA
Christos.Graikos@cwu.edu

Qi Wang
School of Engineering and Computing
University of the West of Scotland
Paisley, UK
Qi.Wang@uws.ac.uk

***Abstract*— Object detection systems mounted on Unmanned Aerial Vehicles (UAVs) have gained momentum in recent years in light of the widespread use cases enabled by such systems in public safety and other areas. Machine learning has emerged as an enabler for improving the performance of object detection. However, there is little existing work that has studied the performance of the machine learning approach, which is computationally resource demanding, in a portable mobile platform for UAV based object detection in user mobility scenarios. This paper evaluates an integrated real-world testbed for this scenario, by employing commercial-off-the-shelf devices including a UAV system and a machine-learning-enabled mobile platform. It presents benchmarking results about the performance of popular machine learning and computer vision frameworks such as TensorFlow and OpenCV and the associated algorithms such as YOLO, embedded in a smartphone execution environment of limited resources. The results highlight opportunities and provide insights into technical gaps to be filled to realize real-time machine-learning-based object detection on a mobile platform with constrained resources.**

***Index Terms*—Machine learning, object detection, image processing, mobile platform, UAV**

I. Introduction

Over the last years, Unmanned Aerial Vehicles (UAV) have gained global popularity for different purposes. This increasing market has been driven by various use cases and advanced technological developments in this field where smart solutions are being sought after for tasks that are currently carried out by humans. These tasks are frequently described as tedious, slow and usually requiring a high level of concentration from humans. Therefore, workers demand help in fields such as agriculture, livestock, rescue and survival, military, construction, and so on. Plenty of use cases exist in each area, for instance, in the agricultural line, farmers have started to deploy drones over the crops for rapid and accurate pesticide application.

Even though there are numerous different use cases of direct application, the aggregation of UAVs and video cameras opened a new research field combined with additional capabilities like computer vision, object detection and 3D modeling. This paper focuses on machine-learning-enabled object detection from UAV video streaming on a mobile platform based on a smartphone. In contrast to the conventional server-based platform, the smartphone-based platform is significantly more lightweight and portable, and thus greatly facilitates use cases for mobile workers such as search and rescue missions in a remote area. Meanwhile, a smartphone-based platform has substantial constraints in the computational power that is required for demanding tasks like applications related to video processing, compared with the server-based counterpart.

In computer vision, different areas can be tackled such as image restoration, motion analysis, object classification or object detection. Object detection is a popular topic where a number of algorithms have been developed; however, their performance in a smartphone-based UAV system has not been sufficiently benchmarked and there is a pressing need to do so. The benchmarking presented in this paper establishes a performance evaluation of machine learning models running on a mobile platform connected to a UAV.

Current object recognition or detection technologies, that include complicated models and generally need to be loaded in the systems. Therefore, they are usually not portable. Hence, power and portability are critical deficiencies in use cases involving drones and smartphone mobile platforms. This paper provides a baseline in the performance that is able to be achieved by a smartphone connected to a UAV, attempting to detect objects near-real time. Consequently, our system evaluates a set of relevant metrics including detection timeliness in terms of frames per second (fps), detection accuracy, battery consumption, smartphone temperature etc.

In order to experiment and analyze realistic data for performance benchmarking, a generic testing platform has been created to study different schemes, employing Commercial-Off-The-Shelf (COTS) devices. As shown in Fig. 2, this platform

runs inside a smartphone, which is connected to the controller that manages the drone. Our system is based on the SDK provided by the drone company (DJI) and every component is integrated into it. In addition, to run object detection algorithms, two main frameworks, *OpenCV* and *Tensorflow*, have been evaluated. Furthermore, these frameworks run two deep-learning-based object detection algorithms: *YOLO* and *Tiny-YOLOv3*, which are among the closest-to-real-time methods achieving a reasonable detection performance.

It is worth mentioning that there are several benefits in deploying machine learning outside the drone compared with the alternative in the drone itself. Firstly, hosting and running machine learning outside (at the smartphone) instead of inside the drone offloads the involved computational-intensive tasks from the drone, thereby leading to prolonged battery lifetime so that the drone can fulfill the flying and video streaming mission. Secondly, COTS drones typically do not allow custom built machine learning capabilities plugged into the onboard system, and thus achieving this outside of the drone is often required. Thirdly, a smartphone can conveniently leverage relevant mobile SDKs for the drone to facilitate the integration of the whole system.

Once the system is integrated, a series of experiments are conducted with empirical results gathered. The analysis of the results leads to identification of gaps and bottlenecks in the system. For instance, in the case of bottlenecks, we focus on the execution of the object detection algorithm, and its impact on real-time video streaming.

The remainder of the paper is as follows. Related work is reviewed in section II, which introduces and compares different technologies such as machine learning and computer vision platforms and object detection algorithms. In addition, it specifies some of the gaps in the existing literature based on object detection in UAV schemes. The proposed design and deployment are then presented in section III, which also describes the requirements for our use case in mobile devices and depicts our benchmarking platform to test it. Section III leads to IV where different experiment scenarios generate results to be compared and analyzed. Finally, section V concludes this paper.

II. Related Work

Since UAVs/drones are a popular topic, a large number of publications have been released. Some of the work has aimed to join machine learning platforms with UAV. These publications, including those for object detection, are mainly destined to traffic control [1]. Other previous works cover use cases such as animal control [3], power line detectors [6], Search and Rescue (SAR) systems [5] or fire detection [8]. As we have highlighted before, running an object detection algorithm is a computational-demanding task, and hence, in most of the previous references the authors used servers/computers as powerful devices to run these algorithms thereby achieving good results. These solutions are not supported by our system due to their non-portability. Instead, this paper will promote the use of mobile devices like smartphones. To focus on the most relevant work from numerous publications in this topic, the following review takes a storyline approach. This has been done by emphasizing the related work in each of the different steps of the process carried out to make technical decisions for building the testbed and experiments (see Fig. 1).

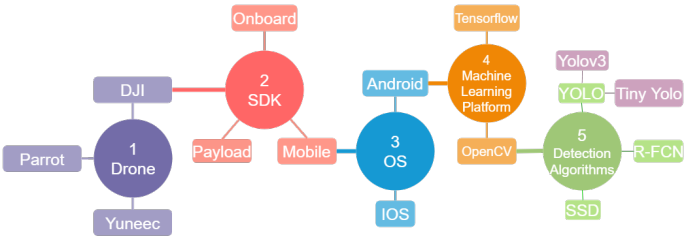

Fig. 1: Diagram of Technologies decision

First of all, there are different brands of drones on the market such as Parrot, Yuneec and DJI. All of those provide SDKs to the costumers for development; however, even though every brand has good quality, **DJI** has been our choice due to the consolidation and dominance they have in the market and the provision of drones for specific purposes. Once the UAV platform is chosen the second step in Fig. 1 presents a choice from different SDKs provided by the company. The Payload SDK is mainly used to integrate payloads in the drone. The Onboard SDK is executed inside the drone, providing software and hardware capabilities leading to e.g., low latency yet higher battery consumption. The Mobile SDK allows a developer to create an App to communicate with the drone. In this work, the **Mobile SDK** is chosen after considering that the drone battery will not be affected (the video is streamed to the controller anyway) and also since the mobile device has a more powerful CPU and GPU than the drone.

In the proposed use case, portability is an important factor. Therefore, a smartphone is employed. As shown in Fig. 1, in the third step, the Mobile SDK provided by DJI can be integrated into IOS or Android. In [6] the authors use the same DJI Mobile platform in an iPad Air and IOS is the operating system chosen. In our use case, **Android** has been selected since it is an open source project, which can be installed in almost every device, and it is easy to modify for our purposes.

In order to run the algorithms, existing work adopted different platforms, particularly Matlab [4] and OpenCV [1] [7]. In our use case, two platforms have been considered and integrated into Android (Fig. 1, the fourth step). OpenCV and Tensorflow are proven platforms for machine learning purposes. Whereas OpenCV is a mature and stable computer vision library, we will focus on benchmarking rather than on creating a novel framework. On the other hand, Tensorflow is a framework that enables a developer to create his/her own algorithms, in addition to the ones it has implemented. Meanwhile, Tensorflow is less mature than OpenCV. In light of the existing algorithms to be integrated and the maturity of the platform, **OpenCV** is selected in our system.

Now the most appropriate object detection algorithm should be chosen (Fig. 1, fifth step). In response to the require-

TABLE I: Comparative analysis of uses cases in the literature

Ref	Aspects						
	Objective	***Algorithm***	***Execution Environment***	***Platform***	***Accuracy***	***FPS***	***Model Size***
[1]	Car & Roads	Viola-Jones	Computer	OpenCV	82.17%	0.94	NG
[2]	People	HAAR-LBP & HOG	NG	NG	73%	NG	NG
[3]	Wild Animals	AlexNet & Proposed	Computer (GTX980TI)	NG	F1 score=0.66	72.65	NG
[4]	Traffic	Optical Flow	Computer	MATLAB	99.8%	NG	NN
[5]	Victims Avalanches	GoogLeNet & SVM	Computer	NG	94.29%	0.9	NG
[6]	Power Line	Canny & Edge Detection	iPad Air	DJI	NG	"Real Time"	NN
[7]	Human Body	Haar-Like	Pentium III	OpenCV	NG	NG	NG
TP	Common Objects	YOLOv3/Tiny-YOLOv3	Smartphone	DJI+OpenCV	55.3/33.1 mAp	0.08/1.37	248/35.4 MB

NG = Not Given; NN = Not Necessary; TP = This Paper; Red = lack of information; Green = uses CNN or portable device

ments of our use case, candidate algorithms need to have three features: speed, accuracy and lightness. In the literature, existing work usually employ servers/computers to run the detection systems. Therefore, lightweight systems are not required since portability is not a priority. In spite of that, existing approaches explore consolidated algorithms such as AlexNet [3], Viola-Jones [1], GoogLeNet [5] and HAAR-LBP [2]. Viola-Jones and HAAR-LBP are based on Haar Feature selection and use cascading classifiers, resulting in significant speed-ups yet lower accuracy than others. AlexNet (plus a proposed metaarchitecture) and GoogLeNet (plus SVM metaarchitecture) are Convolutional Neural Networks (CNN), which are slower and perform heavier tasks but are much more accurate. Accuracy is an important measure to take into account given that drones usually fly at high altitudes. The higher the drone flies, the smaller the object becomes and thus the more difficult it is to be detected. Based on these considerations, in our system we adopt an strategy relying on CNNs to detect objects.

TABLE II: CNN Summary Comparative [9]

CNN	Performance		
	Train	***mAP***	***FPS***
SSD321	COCO Trainval	45.4	16
R-FCN	COCO Trainval	51.9	12
YOLOv3	COCO Trainval	55.3	35
Tiny-YOLOv3	COCO Trainval	33.1	220

Even though in previous works AlexNet and GoogLeNet are used with object detection metaarchitectures, we have studied others algorithms such as Single Shoot Multibox Detection (SSD), You Only Look Once (YOLO) [9] and Region-based Fully Convolutional Networks (RFCN). In Table II there is a comparison among them, trained with the COCO dataset and tested in the same environment (Pascal Titan X). As can be noted from Table II, all of them have favorable mean Average Precision (mAP) but only YOLOv3 achieves real-time detection because it can run at 35 fps. Another strength in YOLOv3 is its tiny version, which is lighter, smaller(less layers) and faster (execution time). Even with less accuracy than other CNNs, it is convenient for constrained environments. Therefore, **YOLOv3** and **Tiny-YOLOv3** are the object detection algorithms proposed for testbed and use case.

In order to highlight the contributions of this paper, a comparative analysis shown in Table I is conducted, comparing related work in the literature with our work in this paper. Owing to the existence of a wide range of use cases, each work is focused on a different topic and applies different algorithms. Despite the good solutions published, there are some gaps to highlight. For instance, the execution environment is usually a server/computer (with the exception of [6]), and those proposals are not portable (Table I). In other work [7], the accuracy of the algorithms is not detailed and neither is the achieved frames per second, thus it is impossible to know whether the algorithm is applicable in real-time or not. Moreover, when mobile devices were employed in previous literature, neural networks were not utilized since faster computer vision techniques are usually considered more suitable for smartphones. In other reference [5], the video was processed once it has been recorded instead of applying the object detection algorithm in real time. Finally, unlike any other approach in the literature, our main contribution is to use a CNN running in a smartphone while it is receiving video streaming from the UAV and benchmark this scheme.

III. Design and Deployment of the Benchmarking Testbed

The purpose of this research is to develop and evaluate a system where a mobile device runs a deep machine learning algorithm for object detection that receives video streaming from a drone. Firstly, our system must follow basic requirements for the deployment of a benchmarking testbed. For instance, it must allow different types of drones, at least from the same brand. Therefore, we employ the SDK provided by DJI. Moreover, the scheme must obtain each frame from the Mobile SDK in a standard format in order to forward it to independent algorithms to be processed. In addition, the machine learning platform should accept the execution of different object detection algorithms, allowing flexibility in order to permit switching between algorithms in a straightforward manner.

Our system consists of three primary entities with different functionalities:

- The drone is connected to the controller through a radio link protocol called *Lightbridge* or *OcuSync*, created by DJI, and while it is flying, the live video is encoded

on board and streamed to the controller. In addition, the drone also receives diverse commands from the controller and simultaneously responds with information collected from the onboard sensors.

- The controller is connected to the drone, which is managed by the pilot. It also has a USB port where the mobile device (smartphone) is connected. In addition to managing the drone, the controller also has the task to forward the video streaming to the mobile device and send commands to the drone from the mobile device.
- The mobile device in this use case is an Android smartphone, which executes, next to the DJI SDK, a machine learning algorithm to detect objects in every frame that receives from the controller. It also shows on the screen the video and where the detected objects are.

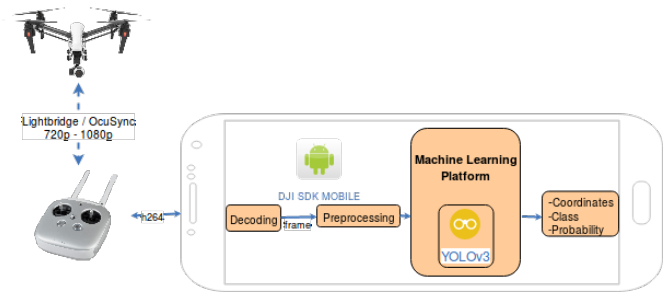

Fig. 2: System Design

As shown in Fig. 2, the live video recorded by the drone is transmitted to the controller through radio and is encoded in H.264. The video resolution transmitted is 720p or 1080p, although it can vary depending on the achievable frame rate and the channel quality. The smartphone is connected by cable to the controller and also receives the video, which needs to be decoded to obtain the YUV frames.

The smartphone receives the video streaming until the objects are detected and shown on the screen, and the whole process is detailed in Fig. 3. In a first step, the Mobile SDK must be authorized by DJI. Subsequently, it will connect to the controller. The next step is to load the OpenCV library, and once that is loaded in memory, the application can run the YOLOv3 algorithm. The model chosen must be one previously trained with YOLOv3. Loading the model is a heavy task that could take some time depending on the size of the model. For instance, if we use a Tiny-YOLOv3 model, the smartphone will take significantly less time to load it. Additionally, it is necessary to load the Neural Network configuration that specifies the number of layers and classes and the input size, among other parameters.

At this point, the smartphone starts receiving video streaming from the drone and decodes each frame to a YUV one. This process executes continuously. Each frame is decoded before sent to the model trained by YOLO. The object detection for each frame is the most time-consuming task because the OpenCV platform is running the complete neural network. Once the detection is finished, the application will ask for

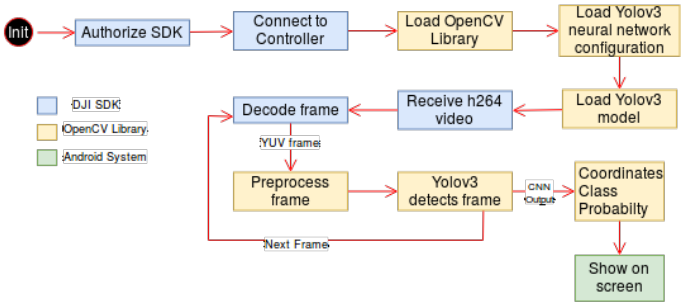

Fig. 3: Application Process

another YUV frame while the CNN output is shown on the screen. This output is the coordinates where the objects are, the class that belongs to and the probability to be that object.

In our approach, we have employed an Inspire 1 model created by DJI. In addition, the connected camera is a ZENMUSE X3, which has a wide angle lens and can record 4K (@25fps), 1920x1080 (@60fps) videos and take 12MP photographs. The remote controller used is the one sold with the drone.l. It also uses DJI Lightbridge technology, which is able to receive video from the drone up to 5 kilometers far away.

The mobile device employed is a Samsung Galaxy S7 Edge (SM-G935F) running Android 7.0, with a Samsung Exynos 8 Octa 8890 with 2.3Ghz Quad-Core and 1.6Ghz Quad-Core, RAM 4G and a battery capacity of 3600mAh. The application developed includes the Mobile SDK version 4.4.1 and the OpenCV Library version 4.0.0, which includes YOLOv3.

The models used for the following experiments are trained by YOLO's creators and are published in their webpage *pjreddie.com*. We employ YOLOv3 and Tiny-YOLOv3 models, which are able to detect 80 different classes as they have been trained with the COCO dataset . These models also have the same input size for images that is 416x416 pixels.

IV. Experiments and Results

Every technology chosen in section II is applied accordingly to the design described in section III. In order to test our approach, different experiments were executed for two schemes: one with YOLOv3 and the other with Tiny-YOLOv3. Then, the power of the tiniest version in constrained environments will be assessed against its normal version. The performance metrics to be measured in the diverse benchmarking tests executed by type of neural networks are frames per second, accuracy, battery consumption, temperature, RAM memory, model size, FLOPS (FLoating point Operations Per Second) and time to load each model.

Comparing Models

Both models are trained and tested with the COCO dataset. In addition, both models also detect the same type of classes which are 80 typical types of objects such as a person, bicycle, car, cat, dog, etc.

However, due to their different purposes, the models have huge differences, as listed in Table III. Firstly, the normal

version has a size of 248MB, whilst the alternative tiny one just has 35.4MB as a more versatile model for constrained environments. This difference in size will lead to divergence in the performance of the device as the following experiments will show. Indeed, more than 200 MB of difference is the consequence of the neural network configuration, since instead of having 75 convolutional layers as YOLOv3, the tiny version just has 9 layers. Secondly, this difference in the configuration of the network leads to an increase in the FLOPS by 91.5% in the YOLOv3 version compared to the tiny one. Finally, the difference in size will also lead to a difference in the time to load each neural network model in the device. In the case of YOLOv3, it takes an average of 1474.8 ms, unlike Tiny-YOLOv3, which takes 156.6ms that is an 89.4% time reduction.

TABLE III: Model Comparative

Model	*Train*	*Classes*	*Size*	*FLOPS*	*Load Time*
YOLOv3	COCO	80	248MB	65.86Bn	1474.8ms
Tiny	COCO	80	35.4MB	5.56Bn	156.6ms

Accuracy

In addition to the previous differences between the models, accuracy also needs to be considered. After training each model with the COCO dataset, the model is sent to a server that is provided by COCO to be tested. Once it has been tested, the results are given in mAP. AP score is given by the average value of the precision in every recall value. In addition, for COCO datasets, mean AP is averaged over all 80 classes and in 10 IoU thresholds.

In this case, YOLOv3 has obtained 55.3 mAP, unlike Tiny-YOLOv3, which has obtained 33.1 mAP. As it can be appreciated, this difference in the accuracy between both models is the consequence of the different neural network configurations. The tiny model is significantly simpler in order to be applicable in constrained environments.

Frames per Second

The speed to run the object detection algorithms is assessed by measuring the number of processed frames per second (fps). In our tests, we have measured the times spent since the frame enters the processing pipeline until we receive the output from it. Fig. 4 shows the instantaneous inverse fps over time, and their cumulative average (solid curves).

As it can be appreciated in Fig. 4, a great difference between both models exists. Both tests are executed until 1000 frames are processed so that an accurate average is obtained. On the one hand, YOLOv3 yields an average of 0.08 fps, and therefore this model causes a delay in the video, leading to non-achievement of a real-time detection.

On the other hand, Tiny-YOLOv3 completes 1000 frames reaching an average of 1.37 fps, which is less than a second per frame. This is not real-time detection either although the performance is improved compared to YOLOv3. Even with these differences, there are still some patterns in the

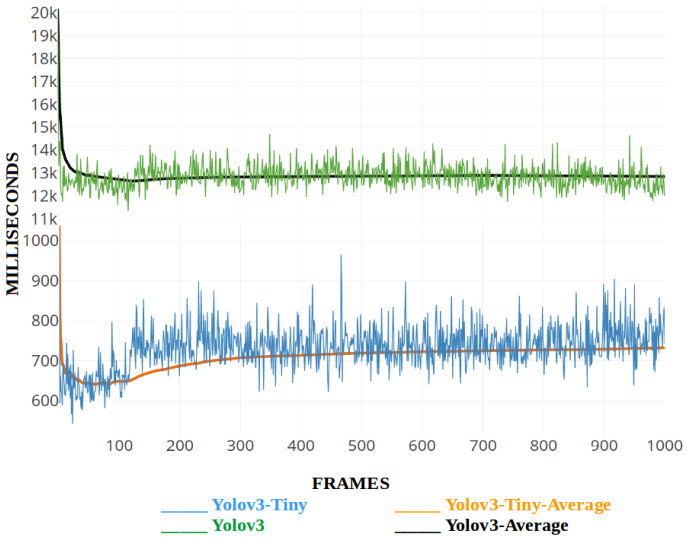

Fig. 4: Milliseconds per Frame

behaviour. In both models, the first frame to be processed requires more time than the others, since the algorithm needs access to the primary memory. However, when the first frame is loaded, the speed increases due to the model being loaded in memory layers near the processor, for instance, a cache. Another similarity is that as the number of frames increases, the performance for each model decreases slightly.

Temperature

Temperature is another key factor to be tested in our system since running a neural network could lead to a considerable increase in the device temperature. In normal environments, a Samsung S7 Edge has a temperature around 30°C although it could reach a peak of 80-90°C. Nevertheless, the smartphone should not be damaged from overheating, which is achievable through shutting down.

Two tests have been conducted for YOLOv3 and Tiny-YOLOv3 in order to control the internal temperature. These tests lasted one hour while running the neural networks. As it is reflected in Fig 5, over the first 1,200 seconds the temperature increases dramatically until it reaches 41°C

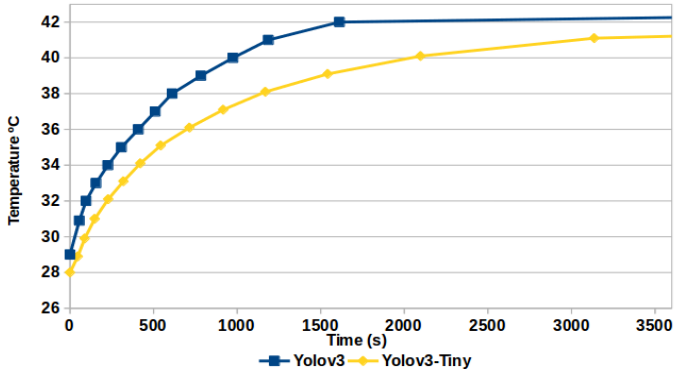

Fig. 5: Temperature

for YOLOv3 and 38°C for Tiny-YOLOv3. This increase is sharper in the YOLOv3 model due to the fact that it performs tasks demanding higher computational power. Immediately, the temperature stabilizes over 40°C for both cases although the tiny model always obtains lower values.

RAM Memory

The size and capacity to run each model depends on the quantity of RAM memory needed. Our device has 4GB of memory available for applications.

After one hour of testing each model, the tiny model consumed an average of RAM of 255MB. Subsequently, the YOLOv3 1.1GB, which is more than four times the memory used by the tiny YOLO.

Over this hour, YOLOv3 reached up to 2.1GB, which is more than a half the device's available memory. This can be compared with the 514MB required by the Tiny-YOLOv3 3 model at its peak use.

Battery Consumption

The high temperature of the device and the amount of RAM memory used by the application leads to a high consumption of the battery because, as we have indicated, running neural networks is a computational-intensive task for small devices. Our mobile device has 3528mAh of battery, and after one hour of detections, the battery consumption for both models has decreased moderately to 3024mAh for YOLOv3 and 3060mAh for Tiny-YOLOv3. In Fig. 6, both lines drop gradually and similarly but the tiny model always consumes less battery.

In addition, it is necessary to mention that the drone's controller charges the smartphone while it is being used, which causes a slower drain of the battery and enables better performance. In any case, even with an input voltage of 5V to charge the device, the consumption drains almost 20% of the battery in just one hour. After these tests are analyzed, the above empirical results have validated the hypothesis that using the tiny model is more appropriate for this use case with UAVs.

As in every neural network problem, the trade-off between speed and accuracy is the main bottleneck for the system. Even with the tiny model, the performance achieved in terms of speed is lower than desired, indicating room for performance improvement in future work.

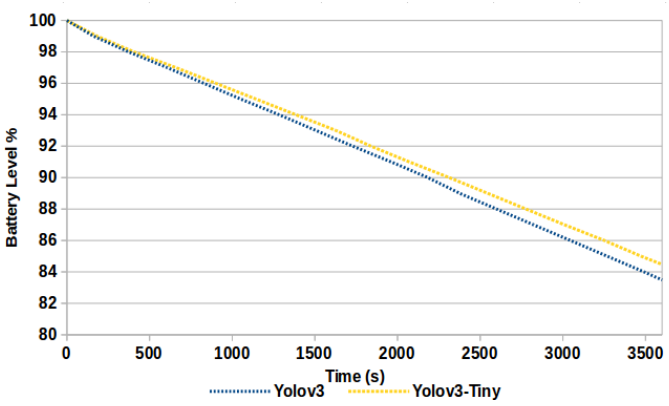

Fig. 6: Battery Consumption

In addition to the accuracy and speed trade-off, the battery consumption is another gap in the scheme. Running these models consumes a significant amount of battery; however, even with this deficiency, it is still a portable system.

V. Conclusion

Empirical performance evaluation of object detection systems on portable platforms linked to UAVs is not readily available in the literature. In light of the little existing work, particularly with portable devices, this paper has established some benchmarking results based on a real-world UAV and smartphone testbed. We have investigated the challenges posed by such resource-constrained environments, and measured and then analyzed the performance in terms of a range of metrics including frames per second, accuracy, battery consumption, temperature, RAM usage, size of the machine learning models, FLOPS and time to load. In particular, we have emphasized the balance between accuracy and velocity in order to obtain an acceptable performance. This has led to the insight that such as a system should not only increase the speed but also reduce battery consumption. Future systems should focus on achieving improved performance with the trade-off between these two factors taken into account, thereby accomplishing real-time object detection whilst gaining battery efficiency.

VI. Acknowledgement

This project is sponsored in part by the CENSIS project *"Smart Unmanned Aerial System for Real-Time Object Detection"* (Grant Ref. CAF440) and the UWS VP Fund.

References

[1] Xu, Y., Yu, G., Wu, X., Wang, Y., & Ma, Y. (2017). "An Enhanced Viola-Jones Vehicle Detection Method from Unmanned Aerial Vehicles Imagery.," IEEE Transactions on Intelligent Transportation Systems, 18(7), 18451856.

[2] Aguilar, W. G., Luna, M. A., Moya, J. F., Abad, V., Parra, H., & Ruiz, H. (2017). "Pedestrian Detection for UAVs Using Cascade Classifiers with Meanshift.," In Proceedings - IEEE 11th International Conference on Semantic Computing, ICSC 2017.

[3] Kellenberger, B., Volpi, M., & Tuia, D. (2017). "Fast animal detection in UAV images using convolutional neural networks.," In International Geoscience and Remote Sensing Symposium (IGARSS) (Vol. 2017July, pp. 866869).

[4] Wang, L., Chen, F., & Yin, H. (2016). "Detecting and tracking vehicles in traffic by unmanned aerial vehicles.," Automation in Construction, 72, 294308.

[5] Bejiga, M. B., Zeggada, A., & Melgani, F. (2016). "Convolutional neural networks for near real-time object detection from UAV imagery in avalanche search and rescue operations.," 2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS), 693696.

[6] Zhou, G., Yuan, J., Yen, I.-L., & Bastani, F. (2016). "Robust real-time UAV based power line detection and tracking.," Image Processing (ICIP), 2016 IEEE International Conference On, 744748.

[7] Rudol, P., & Doherty, P. (2008). "Human body detection and geolocalization for UAV search and rescue missions using color and thermal imagery.," In IEEE Aerospace Conference Proceedings.

[8] Martnez-de Dios, J. R., Merino, L., & Ollero, A. (2001). "Fire Detection Using Autonomous Aerial Vehicles With Infrared and Visual Cameras.," Control, 16(1), 660665.

[9] Redmon, J.,& Farhadi, A. (2018). "YOLOv3: An Incremental Improvement."